Improving Search on Gnutella-Like P2P Systems

Qi Zhao, Jiaoyao Liu, and Jingdong Xu

Department of Computer Science, Nankai University,
Tianjin 300072, China
{qizhao6688,ljy,xjd}@mail.nankai.edu.cn

Abstract. Gnutella has many weaknesses. Queries are handled identically no matter how popular the querying objects are. Blind flooding causes a great amount of redundant messages. In this paper, we explore how to retain the simplicity of Gnutella, while addressing its inherent weaknesses. We propose GPP, a content location solution in which multiple proxy peers are adopted to help a source peer locate desired objects. The underlying philosophy is to route queries to the areas where more results can be found. Towards this end, peers in GPP collaborate to achieve better search performance. Simulation results show that GPP outperforms Modified BFS and s-APS.

Keywords: Unstructured P2P, Search, Efficiency.

1 Introduction

Improvements to Gnutella's flooding mechanism have been studied along three dimensions: blind search, guided search, and group-based search. GPP (Gnutella with Proxy Peers), the search mechanism proposed in this paper, lies in the category of guided search mechanisms. Unlike most of the guided search mechanisms, GPP does not maintain state of the outgoing links or immediate neighborhood. Each peer maintains the performance of its proxy peers, which are distributed over the whole network. The search process of GPP is adaptive such that a targeted performance level can be achieved by the search.

In a guided search, peers locally store metadata that assist in the query process. In DBFS [1], queries are forwarded to a set number of neighbors that are more likely to return successful results. Intelligent BFS [2] uses peer ranking mechanism to forward queries to "good" neighbors. Local Indices [1] and Routing Indices [3] improve search performance via indexing objects stored at other peers. The forwarding process in APS [4] is probabilistic based on feedback from previous searches. In Popularity-Biased Random Walks [5], each step of the random walks is determined based on the content popularities of current neighbors. The authors of [6] introduce a number of local search strategies that utilize high degree peers. [7] proposes an adaptive search mechanism that uses estimate of popularity of a resource in order to choose the parameters of random walk.

2 System Design

Every peer in the system can ask any other peer to be its proxy peer. When a new peer joins the system, it does not know any peer except its neighbors. Therefore, it first

attempts to get proxy peers from its neighbors. The communication process consists of three steps. First, the new peer sends a *Ping_Proxy* message to its neighbors. Next, each neighbor selects a set number of proxy peers from its proxy list. It then sends a *Pong_Proxy* message back to the new peer, which contains the IP addresses of the selected proxy peers. The *Ping_Proxy* and *Pong_Proxy* messages are modified from the *Ping* and *Pong* messages in Gnutella 0.6 protocol [8].

A peer P 's proxy list consists of two parts: a *Primary Proxy List (PPList)* and a *Secondary Proxy List (SPList)*. Each item in the proxy list is the IP address of a proxy peer. When P receives a *Pong_Proxy* message, it first extracts the IP addresses of the proxy peers, and then inserts them into its *PPList*. If its *PPList* is full, these IP addresses will be inserted into its *SPList*. Since P knows the IP addresses of its proxy peers, it can directly communicate with them.

PPList and *SPList* have different roles. When P initiates a query, it first forwards the query message to some items in its *PPList*. P will visit *SPList* only if all the items in its *PPList* have been used and the search is still unsuccessful. The items in *SPList* can enter *PPList* if they outperform the items in *PPList*. Through competition the best items are selected to stay in *PPList*. Obviously, *SPList* is used to memorize the IP addresses of many other peers. It gives P more choices when *PPList* is not enough for a search. *PPList* should be small and fresh. Contrarily, as the backup of *PPList*, *SPList* should be large. It is unnecessary and costly to maintain *SPList* fresh.

In GPP, P maintains its *PPList* fresh by periodically executing the following procedures. P first selects an item Q from its *PPList*, and then sends a *Ping_Proxy* message to Q . If Q does not respond, P will know that Q has left the system. In that case, P will delete Q from its *PPList*. If Q is alive, it will respond with a *Pong_Proxy* message, which contains some items selected from Q 's *PPList*. P will insert these items into its *PPList* with certain probability. The remaining items will be inserted into P 's *SPList*. Hence, peers can share proxy peers with each other. If P is not a proxy peer of Q , Q will insert P into its *PPList* or *SPList* with probability. These procedures will continue until P leaves the system.

2.1 Query Process

When a source peer S initiates a query, it forwards the query messages to some of its proxy peers. On receiving the query message, a proxy peer X initiates a local search on behalf of S . This local search starts a MBFS with dynamic fraction parameter f_h at different hops h when $h \leq n$. For $h > n$, it switches to the Random Walks. This combination of MBFS and Random Walks can greatly cut down redundant query messages [9].

Another important scheme of GPP is *Adaptive Termination*. To explain our idea, we first list the symbols used in this section. N_{mf} denotes the maximum number of proxy peers to query in a search process. N_{rd} denotes the number of results desired. $S_{ppl}(S_{ppl} \leq N_{mf} - 1)$ denotes the size of *PPList*. If S decides to forward a query to K ($1 \leq K \leq N_{mf}$) proxy peers each time, then what is an appropriate value for K ? If $K = N_{mf}$, the maximum number of queries will be initiated simultaneously. Therefore, we call this search scheme *Parallel Search*. It means that S wants the shortest response time. If $K = 1$, S will query its proxy peers one by one, until N_{rd} results have been received.

It shows that S aims to minimize redundant query messages. If $1 < K < N_{mf}$, the search process will be a trade-off between response time and query traffic. Given $1 \leq K < N_{mf}$, S will stop querying its proxy peers if N_{rd} results have been received. The search process is controlled by the source peer S , and it can be terminated adaptively. Therefore, we call this scheme *Adaptive Termination*.

In an *Adaptive Termination* search, the source peer S first selects the items in its $PPList$. If all the items in its $PPList$ have been selected and S has not received N_{rd} results, it will select more proxy peers from its $SPList$. If S has sent a query to an offline proxy peer, it will select another proxy peer to forward the query. This process continues until S has sent the query to an online proxy peer. If S determines to initiate N_{mf} queries simultaneously, it will use all the items in its $PPList$ and select $N_{mf} - S_{ppl}$ items from its $SPList$.

2.2 Ranking Proxy Peers

Since $S_{ppl} < N_{mf}$, $SPList$ is visited in every *Parallel Search* process. Hence, items in $PPList$ can be compared with those in $SPList$ after each process. The performance of proxy peers are evaluated in terms of *Query Hits*. The worst performer in $PPList$ will be deleted, and the best performer in $SPList$ will take its place.

The ranking process in *Adaptive Termination* search is a little complex. If the querying object is popular, the visited proxy peers are likely to be only within the $PPList$. In such cases, we are unable to compare $SPList$ items with $PPList$ items. Therefore, we need another $PPList$ update strategy. We describe the strategy through the following example.

Peer X 's $PPList$ is $\{P_0, P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8\}$. It is divided into two segments: $\{P_0, P_1, P_2, P_3, P_4\}$ and $\{P_5, P_6, P_7, P_8\}$. X initiates an *Adaptive Termination* Search. It receives N_{rd} *Query Hits* after querying l proxy peers.

If $1 \leq l \leq 5$, the queried proxy peers are within the first segment. Take $l = 3$ for example. The queried proxy peers are P_0, P_1 , and P_2 . Assume the *Query Hits* rank is $\{P_2, P_0, P_1\}$. Because P_1 has the lowest rank, it is transferred to the second segment, and P_5 is promoted to the first segment. The updated $PPList$ is $\{P_2, P_0, P_3, P_4, P_5, P_1, P_6, P_7, P_8\}$.

If $6 \leq l \leq 9$, both segments will be visited. Take $l = 7$ for example. The queried proxy peers are $\{P_0, P_1, P_2, P_3, P_4\}$ from the first segment and $\{P_5, P_6\}$ from the second segment. Assume the *Query Hits* rank is $\{P_4, P_0, P_3, P_5, P_1, P_2, P_6\}$. P_6 , an item in the second segment, has the lowest rank. It will be eliminated from $PPList$, and an item from $SPList$ will be inserted into $PPList$. The updated $PPList$ will be $\{P_4, P_0, P_3, P_5, P_1, P_2, P_7, P_8, P_{SPList}\}$. There is another case, in which the *Query Hits* rank is $\{P_4, P_0, P_3, P_5, P_1, P_6, P_2\}$. P_2 , an item in the second segment, has the lowest rank. The $PPList$ will be updated into $\{P_4, P_0, P_3, P_5, P_1, P_6, P_2, P_7, P_8\}$.

In the other cases ($10 \leq l \leq N_{mf}$), the worst performer in $PPList$ will be deleted, and the best performer in $SPList$ will be inserted into $PPList$. $PPList$ will be updated according to the *Query Hits* rank.

3 Conclusion and Future Work

In this paper, we proposed GPP, an efficient search mechanism in unstructured P2P networks. The underlying philosophy is to route queries to the areas where more results can be found. Towards this end, peers in GPP collaborate to achieve better search performance. We discussed a variety of design issues and trade-offs in implementing GPP. As future work, we plan to give more formal analysis to GPP and other guided search methods through mathematical modeling.

References

1. B. Yang, H. Garcia-Molina: Improving Search in Peer-to-Peer Systems, *In Proceedings of ICDCS*, (2002).
2. V. Kalogeraki, D. Gunopulos, D. Zeinalipour-Yazti: A Local Search Mechanism for Peer-to-Peer Networks, *In Proceedings of CIKM*, (2002).
3. A. Crespo, H. Garcia-Molina: Routing Indices for Peer-to-Peer Systems, *In Proceedings of ICDCS*, (2002).
4. D. Tsoumakos, N. Roussopoulos: Adaptive Probabilistic Search (APS) for Peer-to-Peer Networks, Technical Report CS-TR-4451, Univ. of Maryland, (2003).
5. M. Zhong, K. Shen: Popularity-Biased Random Walks for Peer-to-Peer Search under the Square-Root Principle, *In Proceedings of IPTPS*, (2006).
6. L. A. Adamic, R. M. Lukose, A. R. Puniyani, B. A. Huberman: Search in Power-Law Networks, *Physical Review E*, 64, (2001).
7. N. Bisnik, A. Abouzeid: Modeling and Analysis of Random Walk Search Algorithms in P2P Networks, *In Proceedings of Second International Workshop on Hot Topics in Peer-to-Peer Systems*, (2005).
8. The Gnutella protocol specification 0.6, <http://rfcgnutella.sourceforge.net>.
9. H. Wang, T. Lin: On Efficiency in Searching Networks, *In Proceedings of IEEE INFOCOM*, (2005).